REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-01-
0206

Public reporting burden for this collection of information is estimated to average 1 hour per response, includi ces,
gathering and maintaining the data needed, and completing and reviewing the collection of information. Sen this
collection of information, including suggestions for reducing this burden, to Washington Headquarters Servic 'son
Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Pape

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPO 1 Apr 97 - 30 Jun 00
4. TITLE AND SUBTITLE Exploiting Problem Structure in Scheduling		5. FUNDING NUMBERS F49620-97-1-0271
6. AUTHOR(S) Adele Howe		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Science Department, Colorado State University, Fort Collins, CO 80524		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR, 801 N. Randolph Street, Room 732, Arlington, VA 22203-1977		10. SPONSORING/MONITORING AGENCY REPORT NUMBER F49620-97-1-0271
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release.		12b. DISTRIBUTION CODE

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFOSR)
NOTICE OF TRANSMITTAL DTIC. THIS TECHNICAL REPORT
HAS BEEN REVIEWED AND IS APPROVED FOR PUBLIC RELEASE
LAW AFR 190-12. DISTRIBUTION IS UNLIMITED.

13. ABSTRACT *(Maximum 200 words)*

Many scheduling algorithms have been developed, but surprisingly there is little guidance as to which method to use for a given application. This project addressed this gap be-tween the science and practice of scheduling by developing a methodology for assessing performance and applying it to two problems: scheduling a manufacturing flow shop and scheduling access requests for the Air Force satcllite control network. We found flaws in the conventional practice of evaluating scheduling algorithms. In particular, performance on popular benchmark problems does not generalize to problems with realistic problem struc-ture. Additionally, we found that the performance of algorithms previously deemed to be state-of-the-art may be dominated by much simpler and computationally faster algorithms. We were able to explain the differences in performance by characteristic patterns in the search spaces of structured problems.

14. SUBJECT TERMS			15. NUMBER OF PAGES 28
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Final Report for AFOSR #F49620-97-1-0271
Exploiting Problem Structure in Scheduling
April 1, 1997 to June 30, 2000

Adele Howe L. Darrell Whitley
Computer Science Department
Colorado State University
Fort Collins, CO 80524
email: {howe,whitley}@cs.colostate.edu

October 2, 2000

Abstract

Many scheduling algorithms have been developed, but surprisingly there is little guidance as to which method to use for a given application. This project addressed this gap between the science and practice of scheduling by developing a methodology for assessing performance and applying it to two problems: scheduling a manufacturing flow shop and scheduling access requests for the Air Force satellite control network. We found flaws in the conventional practice of evaluating scheduling algorithms. In particular, performance on popular benchmark problems does not generalize to problems with realistic problem structure. Additionally, we found that the performance of algorithms previously deemed to be state-of-the-art may be dominated by much simpler and computationally faster algorithms. We were able to explain the differences in performance by characteristic patterns in the search spaces of structured problems.

20010404 111

Contents

1 Project Objectives

The objective of our project was to exploit the benefits of systematic and non-systematic search techniques for scheduling. To do so, we first needed to better understand the performance of state-of-the-art scheduling algorithms. Many algorithms have been developed, but scientists and developers know little about which algorithms to use for particular applications or even how to improve performance for particular applications. One of the major problems facing developers is determining how a system developed on a given set of problems will actually perform in the field.

To address our objective and the problem of scheduler developers, we proposed to analyze information about the effects of structure in test problems on algorithm performance. Additionally, we proposed to implement a variety of scheduling methods, including our own Path Relinking algorithm, and comparatively evaluate them.

As such, our project pursued three avenues of inquiry: empirical studies of algorithms on scheduling problems, theoretical analyses of search, and application of our findings to an Air Force scheduling problem. Thus, our report is organized in three parts to correspond to each of these avenues.

2 Flow Shop Scheduling Studies

Manufacturing scheduling, in particular flow shop and job shop, has garnered the most attention from researchers and algorithm developers. Most scheduling algorithms have been developed for and tested on these applications. Thus, we started our analysis of scheduling performance on the most basic class of problem: flow shop scheduling.

Our empirical analyses examined key questions: Do algorithms scale-up? Does algorithm performance depend on key problem features, in particular problem structure? How much effort is required to improve solutions? How difficult are the problems?

Most researchers report the performance of their scheduling algorithms on problems available from a well known synthetic benchmark set reputed to contain *difficult* problems. The underlying assumption of studies based on these problems is that if an algorithm performs well on difficult synthetic problems, then it will also perform well on scheduling applications. However, the problems in this test suite were generated by selecting job processing times from a single uniform random distribution; the problems were then submitted to a search algorithm for solution. Problems were accepted as "difficult" if the search algorithm had trouble consistently finding a good solution, as measured relative either to the lower bound or to a best known solution.

However, real-world problems are *not* random — they typically are characterized by some amount of structure, though it is rarely quantified and categorized. Starting from this observation, we questioned the underlying assumptions behind the use of benchmarks and examined two of the previously mentioned questions: Does the performance of state-of-the-art algorithms scale-up to PFSPs (Permutation Flow Shop Problems) that have structural features representative of those found in some real-world problems? How difficult are structured versus random problems?

To address these questions, we constructed a test suite generator based on known characteristics of some real-world problems. We then compared the performance of a suite of state-of-the-art algorithms and heuristics on problems with varying structure. We also analyzed the the search spaces for the problems to explain the observed differences in performance.

2.1 A Structured PFSP Generator

The scheduling problem was the well-known n by m permutation flow-shop sequencing problem (PFSP) [8]. Here, n jobs must be processed, in the same order, on each of m machines. Concurrency is not allowed: a machine can only process a single job at a time, and processing must be completed once initiated. Furthermore, machine $j+1$ cannot begin processing a job until machine j has completed processing of the same job. Each job i requires a processing time of d_{ij} on the jth machine. A candidate solution to the PFSP is simply a permutation of the n jobs, π. Given that the first job in π on the first machine begins at time step 0, the makespan of π is defined to be the finish time of the last job in π on the last machine. The objective is to find a permutation π such that the makespan is minimized.

The most commonly used PFSP benchmark problems are those introduced in Taillard (1993) and available through the OR library. Taillard generated his problems by selecting the processing times d_{ij} uniformly from the interval [1,99] and then choosing a subset of problems based on several criteria, including problem difficulty as measured by a Tabu search algorithm. In expectation, and in contrast to many real-world problems, Taillard's problems contain few or no discernible structural features.

Non-random structure is produced by drawing processing times from a number of Gaussian distributions. For job-correlated problems, we use n distributions, one for each job. For machine-correlated problems, we use m distributions, one for each machine. A parameter α controls the expected degree of distribution overlap. Low degrees of distribution overlap yield PFSPs with more structure because processing times selected from two distributions with little overlap are typically much different. As the degree of overlap increases, the amount of structure decreases; in the limit, the distributions share the same mean, albeit with different standard deviations.

In contrast to Taillard (1993), we do not filter for so-called difficult problems. Instead, we concern ourselves with algorithm performance on *classes* of problems. Taillard defines difficulty relative to a specific algorithm. Thus, any comparison of different algorithms would be biased by such filtering, as problem difficulty can only be defined relative to an algorithm.

2.2 PFSP Algorithms

We implemented state-of-the-art algorithms based on three search methodologies: 1) path relinking, 2) incremental construction, and 3) iterative sampling. The path relinking algorithm by Reeves and Yamada (1998) was selected because it has demonstrated some of the best performance to date on the problems from Taillard's test suite and was similar in both implementation and performance to a "path relinking" algorithm that we had been developing in house. Incremental construction refers to algorithms that use heuristics to build up a schedule; this class was included because excellent heuristics are available for the PFSP domain.

Iterative sampling refers to a class of stochastic algorithms ranging from random sampling to random-starts local search; this class was included primarily because of reported successes of such algorithms on various scheduling applications.

Path Relinking Path relinking is a general search strategy in which the search space is explored by looking for additional optima near two known local optima. During the process of 'linking' or constructing a path between two local optima, the algorithm can check the intervening area for other optima. Path relinking is the basis for the Reeves/Yamada PFSP algorithm [18], which we denote by *pathrelink*.

Incremental Construction Algorithms NEH [13] is widely regarded as the best performing heuristic for the PFSP [20]. The NEH algorithm can be summarized as follows:

```
(1) Order the $n$ jobs by decreasing sums of total job processing times on the machines.
(2) Take the first two jobs and schedule them so as to minimize the partial makespan as
    if there were only two jobs.
(3) For $k$=3 to $n$ do
      Insert the $k$-th job into the location in the partial schedule,
      among the $k$ possible, which minimizes the partial makespan.
```

Despite its simplicity ($O(n^3m)$), NEH produces reasonably good solutions to Taillard's benchmark problems. Solutions produced by *pathrelink* are either competitive with or exceed the previously best known solutions. Yet the comparison is hardly fair: the run-time of NEH is several orders of magnitude less.

We also implemented several systematic backtracking algorithms: Chronological Backtracking (CB), Limited Discrepancy Search (LDS), Depth-bounded Discrepancy Search (DDS), and Heuristic-Biased Stochastic Sampling (HBSS). Chronological backtracking serves as a baseline performer for the heuristic backtracking algorithms. LDS, DDS and HBSS had previously shown promising performance on scheduling problems. For LDS [9] and DDS [23], a discrepancy is defined as any point in the search where the advice of the heuristic is not followed. LDS iteratively increases the maximum number of discrepancies allowed on each path from the root of the search tree to any leaf. In contrast, DDS iteratively increases the depth in the search tree at which discrepancies are allowed. Both algorithms assume the availability of a good heuristic, such as NEH. DDS further assumes that discrepancies required to achieve near-optimal solutions should occur at relatively shallow depths in the search tree. HBSS [3] is an incremental construction algorithm in which multiple root-to-leaf paths are stochastically generated. Instead of randomly choosing a move, an acceptance probability is associated with each possible move. This acceptance probability is based on the rank of the move assigned by the heuristic.

Iterative Sampling Algorithms We also implemented several iterative sampling algorithms. In random sampling, a number of random permutations are generated and evaluated.

Another iterative sampling algorithm can be obtained by modifying step (2) of the NEH algorithm. Instead of selecting the two largest jobs, we instead choose two jobs at random. Step (3) of the NEH is then followed, without backtracking, to produce a complete schedule. We denote this algorithm by *NEH-RS* (NEH with random-starts).

In iterative random sampling, local search is applied to the randomly generated solutions; we denote this algorithm by *itsampls* (iterative random sampling with local search). Following Reeves and Yamada (1998), we use a shift local search operator coupled with a next-descent search strategy. Let π represent a permutation and π_i be the element in the i^{th} position of the permutation. The operation $\pi_i \mapsto \pi_j$ denotes that the i^{th} element in the original permutation is re-mapped to the j^{th} position. Given two randomly selected positions i and j, $i < j$, the shift operator $SH(i,j)$ transforms π as follows:

$$SH(i,j) : \pi \longrightarrow \pi \begin{cases} \pi_k \mapsto \pi_{k+1} & \text{if } i \leq k < j \\ \pi_j \mapsto \pi_i & \\ \pi_k \mapsto \pi_k & \text{otherwise} \end{cases}$$

The operator is applied to all pairs of jobs in a random order, with each improving or equal move accepted.

2.3 Relative Algorithm Performance: Empirical Results

Algorithm performance was measured on six problem classes consisting of three sizes of both job and machine-correlated problems: 50, 100 and 200 jobs, all executed on 20 machines. For each problem class, we generated 100 problem instances with α ranging from 0.1 to 1.0, in increments of 0.1. Varying the problem size allows us to assess algorithm scalability, while varying α allows us to assess the influence of structure on algorithm performance.

For the *pathrelink* algorithm, either local search or path projection is performed at each iteration. Each local search or path projection involves 1000 steps, each requiring an evaluation. The total number of evaluations was limited to 100,000. For all *itsampls* trials, we allowed two 'all-pairs' iterations and limited the total number of evaluations to 100K.

For each algorithm, we recorded the best makespan obtained on each problem. The optimal makespans for these problems are unknown; we measured individual algorithm performance by computing the *percent above the best solution found* by any of the search algorithms considered. Finally, we obtained a summary measure of algorithm performance at each level of α for each problem class by computing the average percent above best for the 100 problems.

Machine-Correlated Problems Figures 1a, 2a, and 3a record algorithm performance on 50 by 20, 100 by 20, and 200 by 20 *machine-correlated* problems. All algorithms significantly outperformed random sampling. As a group, the stochastic algorithms (*itsampls*, NEH-RS, and HBSS) outperform the deterministic algorithms (NEH, LDS, and DDS). The superior performance of HBSS and NEH-RS is both sustained and magnified as problem size increases: both algorithms scale extremely well. The performance of *itsampls* fails to scale to 200 by 20 problems; for small values of α, it is outperformed by the deterministic algorithms. Interestingly, the two strongest performers, HBSS and NEH-RS, are based on a domain-specific heuristic (NEH), while *itsampls* is not.

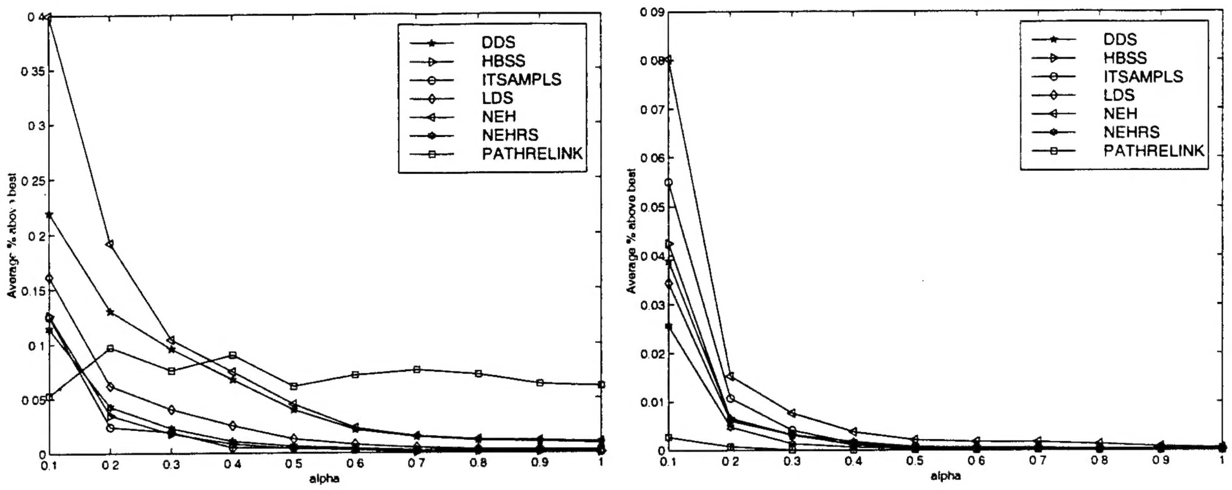


Figure 1: 50x20 machine-correlated (left) and job-correlated (right) results.

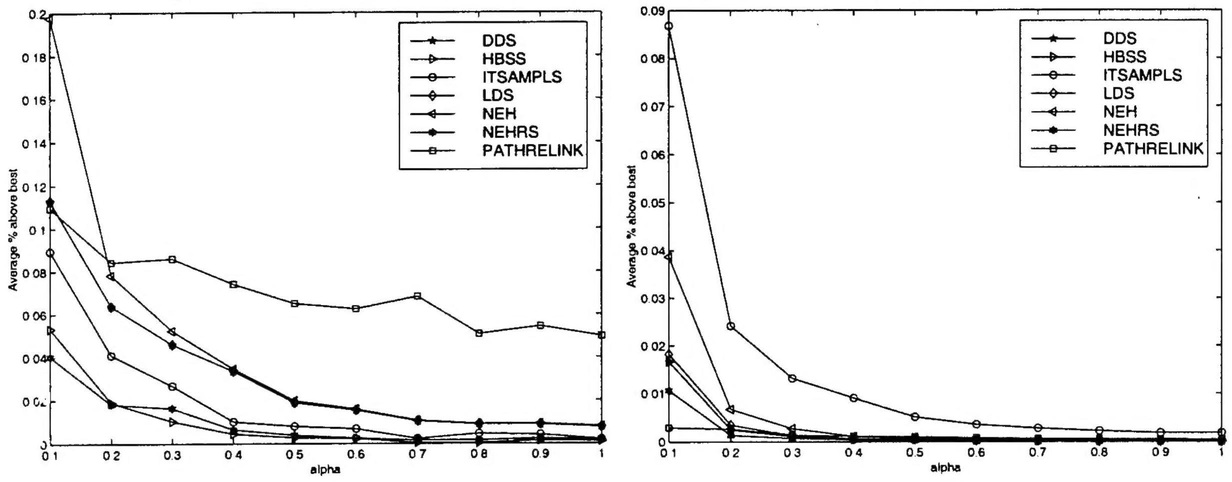


Figure 2: 100x20 machine-correlated (left) and job-correlated (right) results.

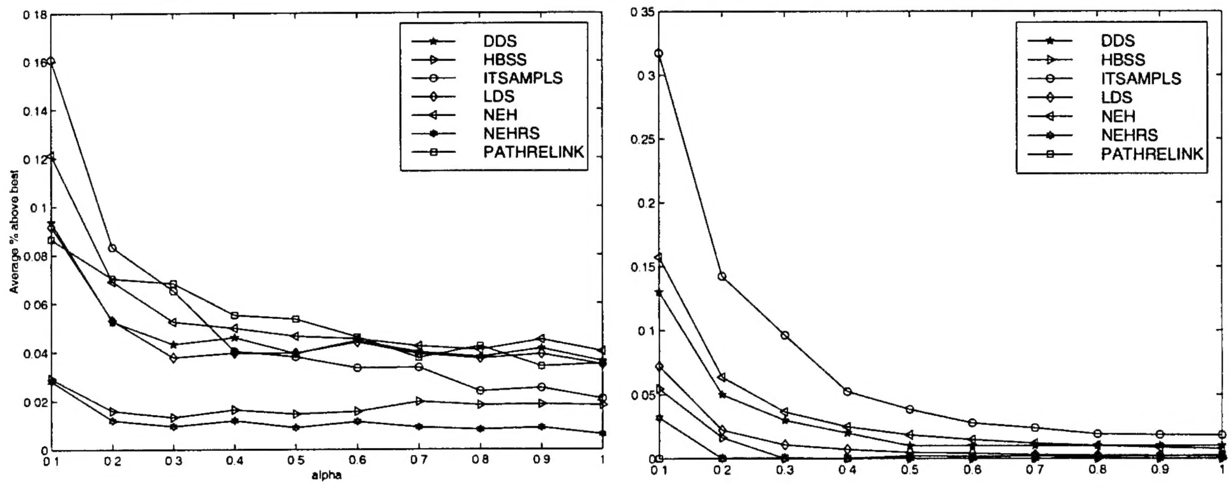


Figure 3: 200x20 machine-correlated (left) and job-correlated (right) results.

The most striking aspect of Figures 1 - 3 is the inconsistent, and often poor, performance of the *pathrelink* algorithm. For the machine correlated problems in Figure 1, *pathrelink* starts to under-perform relative to both HBSS and NEH-RS between α equal to 0.1 and 0.2. At larger values of α, *pathrelink* is outperformed by many of the other, simpler algorithms.

Job-Correlated Problems The rightmost graphs in Figures 1, 2, and 3 record algorithm performance on 50 by 20, 100 by 20, and 200 by 20 *job-correlated* problems. Again, all algorithms significantly outperformed random sampling. As was the case for machine-correlated problems, the stochastic algorithms outperform the deterministic algorithms, excepting the performance of *itsampls*, which fails to scale to larger problem sizes. Here, the performance degradation of *itsampls* is even more rapid than that exhibited on machine-correlated problems; on 100 by 20 and 200 by 20 problems, NEH obtains superior results at *all* values of α.

NEH-RS remains the strongest overall performer; it only slightly under-performs *pathrelink* when $\alpha = 0.1$. The move from 100 by 20 to 200 by 20 problems results in greater differences in algorithm performance, although the relative order remains stable. In contrast to the machine-correlated results, *pathrelink* consistently outperforms all other algorithms, on all problem sizes, with the sole exception of NEH-RS. Even more interesting is the fact the performance *pathrelink* appears to be *independent* of α, the level of non-random problem structure.

2.4 Assessing Problem Structure

The second question addressed by this study is: are the problems hard? In particular, we wanted to determine what it means to be a "hard" problem and how this influences algorithm performance. Taillard's problems have been filtered to be "hard." In other areas of AI, the phase transition regions of problems such as SAT or Hamiltonian circuits have been explored as a source of difficult test instances [6].

Reeves and Yamada (1998) show that Taillard's difficult flow-shop problems display a *big-valley* problem structure when the *shift* local search operator is used. The notion of *big-valley* is somewhat imprecise. It suggests that 1) local optima tend to be relatively close to other local optima, 2) better local optima tend to be closer to global optima, and 3) local optima near one another have similar evaluations.

We hypothesized that the poor performance of the state-of-the-art *pathrelink* algorithm on machine correlated problems might be attributable to a lack of the big-valley structure in our problems. Thus, we tested the different types of PFSPs for it. In our experimental setup, the underlying distribution (Gaussian or uniform) and the parameters defining the type of structure (correlation on jobs, machines, and α, or no correlation) are the independent variables. The dependent variables are the distance between local optima and the quality of the solutions obtained, quantifying the presence and extent of the big-valley structure.

For each problem, we generated 2000 local optima by starting with random permutations and running local search using the shift operator. The shift operator is repeatedly applied, in a next-descent strategy for all the possible pairs of jobs in the permutation, in a random order, until two passes through all the pairs does not result in any improvement. Because the global optima for our problems are unknown, we next computed for each local optimum its average distance to all the other local optima, as was done in the previous study [18].

Taillard's Problems In the leftmost graph of Figure 4, the results for a 50x20 problem (TA052) from Taillard's test suite serve as a prototypical example of a big-valley structure. In this figure, the local optima tend to be clustered, with good local optima close to each other. To determine the impact of the choice of distribution on Taillard's problem generator, we replaced the uniform distribution on the interval [1,99] with the Gaussian distribution $\eta(50.16)$. The rightmost graph in Figure 4 shows a typical example of the resulting scatterplots. The choice of distribution appears to have no significant impact on the existence of the big-valley structure.

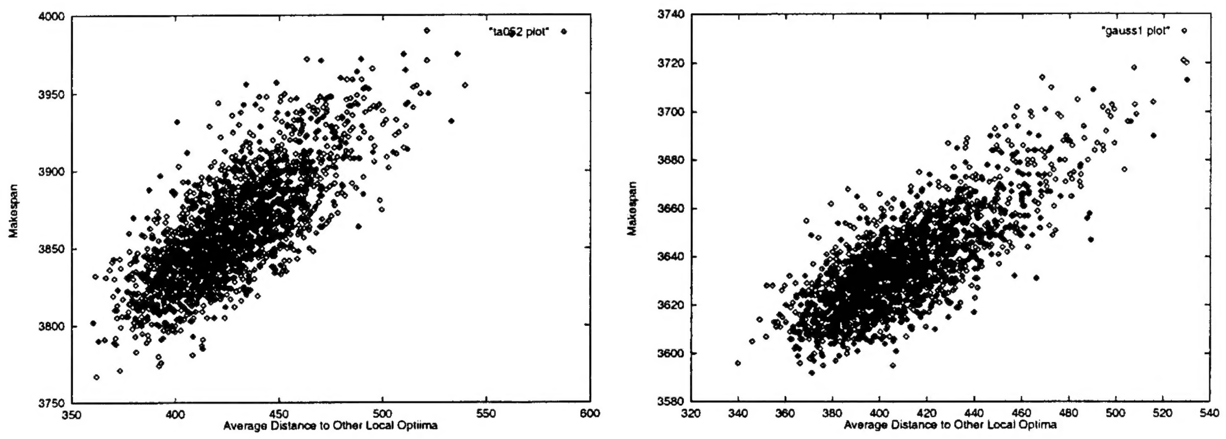
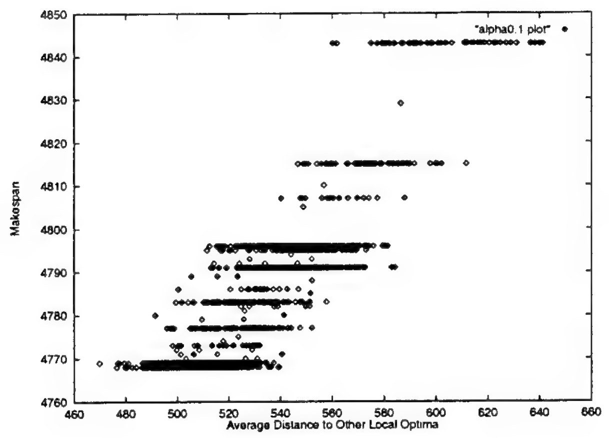


Figure 4: Taillard's TA052 50x20 instance, uniform distribution, no correlation (leftmost) and a Taillard-like Gaussian distribution 50x20 instance (rightmost).

Correlated Problems We next investigated the effect of correlation on the landscape generated by the shift operator, when a Gaussian distribution is used. We generated local optima and distance measures for several 50x20 instances of both job and machine-correlated problems, varying α from 0.1 to 1.0 in increments of 0.1. The graphs in Figure 5 shows the result for a machine-correlated problem generated with α equal to 0.1. The results for job-correlated problems were similar. Note that an α of 0.1 represents a very low level of correlation. While there is still evidence of a big-valley structure, another dominant structural feature begins to emerge: strata of local optima at specific makespan values. Further analysis indicates that many members of the same stratum actually belong to the same plateau and can be partitioned into a small number of distinct local optima.

Although not shown, we also varied the amount of problem structure as measured by α. The empirical evidence suggests that the number of plateaus gradually drops to only a few, and all local optima are gradually absorbed into some plateau.

Finally, we checked for an interaction effect of the distribution and correlation. The question is whether the plateaus emerged due to the combined influence of correlation and Gaussian distribution. We therefore generated job and machine-correlated problems using a uniform distribution. The result from a 50x20 machine-correlated instance (α=0.1) is shown in the leftmost graph of Figure 5; the results for job-correlated instances were similar. As with non-correlated problems, the choice of distribution appears to have little or no impact on the results.



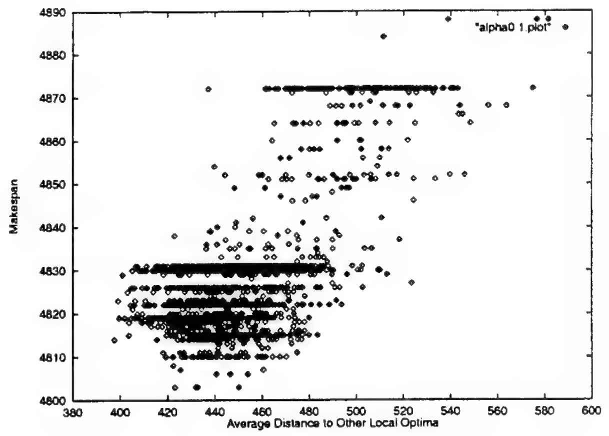


Figure 5: Machine-correlated 50x20 Instances, α=0.1. The left graph is a uniform distribution; the right graph is a Gaussian distribution.

For job and machine-correlated problems, the big-valley structure is not the dominant structural feature of the fitness landscape. As the level of structure is increased, the landscape is dominated by only a few very large plateaus of local optima.

3 Advances to Theory of Search

Another goal of this research was to explore the idea that there is no general purpose approach to optimization. In 1995, this idea was formalized in the "No Free Lunch" theorem [26]. The proof to this theorem shows that over all possible discrete optimization problems, all search algorithms have the same performance. So if an algorithm A is better than algorithm B on a set of benchmarks, then algorithm A is also guaranteed to be worst than B on some complementary set of problems. This suggests that search algorithms must always be application specific to some degree, but it also suggests that good results on benchmarks does not necessarily translate into good results on other problems. However, the set of "all possible discrete functions" is a rather large and nebulous set of problems. Does the No Free Lunch theorem say anything about real world problems? This question has been hotly debated, but there have been no proofs concerning whether No Free Lunch does or does not apply to real problems. Note that this general question is very much relevant to our interest in comparing state-of-the-art algorithms on different classes of problems: do good results on hard benchmarks say anything about performance in general? The No Free Lunch theorem provides good reason for caution.

We have generated two major results related to the "No Free Lunch" Theorem. Our results are as follows.

3.1 No Free Lunch and Problem Description Complexity

First it is relatively simple to show that the No Free Lunch theorem also holds for all finite sets of functions define over N!. Permutations can represent search problems when all points in the search space have unique evaluations. Given a particular set of N evaluations we have N! search algorithms and N! possible functions. We can then prove that the average description length over the set of N! functions must be O(N lg N). Thus if the size of the search space is exponentially large with respect to a parameter set which specifies a point in the search space (as is true for all search problems), then the description length of each member of the set of N! functions must also be exponential on average.

Consider a set composed of N unique values, these N values can be mapped to a set V of evaluations for some objective function. Let X be the set of points and V the set of evaluations; we then define a one-to-one objective function f:

$$f(x) = v, \qquad v \in V, x \in X.$$

Construct a set Π of all permutation over the values in V. In this case, the set Π represents all objective functions which can be constructed over V. We will also use Π to represent all search algorithms which can be applied to the set of evaluation functions which can be constructed over V. We will represent an algorithm by the order in which it samples the values in V. If different search procedures enumerate the points in V in the same order, then they form part of an equivalence class. Resampling of points is ignored. Hence, in this context, algorithms as well as functions are permutations. It is easy to show that a specialization of the "No Free Lunch" theorem result holds [26] [16] [7]. *On average, no algorithm is better than random enumeration in locating the global optimum. If algorithms are executed for only* **m** *steps, every algorithm find the same set of best so-far solutions over all functions.*

Focusing attention on a well defined set of permutations of finite length, allows one to make more detailed comments about the No Free Lunch result as it pertains to this set.

THEOREM : The set of N! functions corresponding to Π have a description length of O(N lg N) bits on average, where N is the number of points in the search space.

Proof: The proof follows the well known proof demonstrating that the best sorting algorithms have complexity O(N lg N). First, since we have N! functions, we would like to "tag" each function with a bit string which uniquely identifies that function. We then make each of these tags a leaf in a binary tree. The "tag" acts as an address which tells us to go left or right at each point in the tree in order to reach a leaf node corresponding to that function. But the "tag" also uniquely identifies the function. The tree is constructed in a balanced fashion so that the height of the tree corresponds to the number of bits needed to tag each function. Since there are N! leaves in the tree, the height of the tree must be O(lg(N!)) = O(N lg N). Thus O(N lg N) bits are required to uniquely label each function. **QED.**

Note that the number of bits required to construct a full enumeration of any permutation of N elements is also O(N lg N) bits, since there are N elements and lg N bits are needed to distinguish each element. Thus, most of these functions have exponential description. To be NP-Complete, the description length must be polynomial. This means that an NP-Complete problem class *cannot* be used to generate all N! functions. This includes NK-Landscapes [10] and MAXSAT, for example.

This is, of course, one of the major concerns about No Free Lunch results. Do "No Free Lunch" results really apply to sets of functions which are of practical interest? Yet this same concern is often overlooked when theoretical researchers wish to make mathematical observations about search. For example, proofs relating the number of expected optima over all possible functions [17], or the expected path length to a local optimum over all possible functions [22] under local search are computed with respect to the set of N! functions. This is the first results that shows that most of the No Free Lunch results hold over sets that do not correspond to real world problems. On the other hand, there are other functions where the number of distinct evaluations is small relative to the size of the search space. So it is still possible that No Free Lunch results holds over sets of functions that are of practical interest. We hope to resolve the remaining open issues related to this question in the near future, but our work has already clarified the relationship of the No Free Lunch theorem to most real-world optimization problems where the number of evaluations is large relative to the size of the search space: No Free Lunch in its general form does not hold for most problems of practical interest.

3.2 No Free Lunch and Representation

A form of the "No Free Lunch" theorem holds for representations. In this case, the "No Free Lunch" theorem is stronger, because one can show that all possible algorithms are the same over only 2 representations. This is true for Standard Binary Reflected Gray Coded representations and Standard Binary Coded Decimal representations. This is troubling because we would like to have guidance about what representation is best on average. Our results in the preceding subsection are not specific enough to throw light on this problem. We next explore this problem in more detail, then show that one can indeed distinguish between these two representations.

Over all possible functions, Gray codes and Standard Binary codes are equal in that they both cover the set of all possible bit representations [25]. In spite of this No Free Lunch result [26], applications oriented researchers have often argued for the use of Gray codes [4]. The debate as to whether Gray coding is better than Binary representations has been a classic example of where theory and practice clash. Our results bring theory and practice closer together and yields new insights into the role of representation during search.

A measure of complexity is proposed that counts the number of local minima in any given problem representation. On average, functions that have fewer minima are assumed to be less complex than functions with more minima.

The set of functions which are considered are such that they can be remapped so that the range of the functions is 0 to $2^L - 1$ and the domain the values 0 to $2^L - 1$. This restriction of the domain and range has the advantage of mapping all functions that can be discretized and represented by an L bit encoding onto a well defined finite set of functions. We will refer to this set of functions as F, and f_j as a function in F.

Two neighborhood structures over F will be defined. Let $\mathbf{F}$ represent the set of wrapped neighborhoods for functions in F. For all integers, i, $i-1$ and $i+1$ are neighbors and the addition and subtraction operations are *mod* $2^L - 1$ so that endpoint are also neighbors.

Let $\mathcal{F}$ represent the set of non-wrapping neighborhoods for functions in F. For all integers, i, $i-1$ and $i+1$ are neighbors except that endpoints have a single neighbor; thus 0 and $2^L - 1$ are not neighbors. This is a relatively minor difference, but one that turns out to be useful for proving properties about Gray and Binary representations.

Typically, when functions are encoded as bit strings, they are first mapped to $\mathcal{F}$ or $\mathbf{F}$. Note this neighborhood preserves local optima in the original function under search operators that move to adjacent points in the space. The neighborhoods $\mathcal{F}$ and $\mathbf{F}$ preserve the connectivity of the original function. After mapping to $\mathcal{F}$ or $\mathbf{F}$, a bit representation is introduced. Assuming that the original function has limited complexity (in that sense that it has fewer than the maximum possible number of local optima), it can be proven that Gray codes on average provide theoretical advantages over standard Binary encodings. In this situation, a special family of Gray codes (called "reflected" Gray codes) will in expectation induce fewer total optima than the corresponding set of Binary representations.

This lead to the following general result.

THEOREM: For classes of discrete parameter optimization problems with a bounded number of local optimal, it can be proven that Gray codes representations induce fewer optima on average than Standard Binary Coded Decimal representations. In this sense, Gray code representations are better on average.

The proof is long and complex and is published elsewhere [24]. The proof also implies that a specific bound on the number of optima that can be induced for a specific problems also has a specific bound under Gray codes, but it can also be proven that a similar bound does not exist for Standard Binary Coded Decimal representations.

This theoretical result is consistent with the experimental results that have accumulated over the last ten year. Again, this is the first major result that shows that one can in general say that one representation is generally better than another over large sets of problems that are of practical interest.

4 Application: Air Force Satellite Scheduling

At the PI meeting in Monterey in May 1998, we were introduced to an Air Force scheduling application: scheduling requests for access to the satellite ground stations. The problem is severely over-constrained: they receive far more requests for access than can be accommodated. At present, the task is accomplished by human schedulers with relatively little computer assistance and requires schedules to be generated about a week in advance. Shortening the scheduling window from one week down to two or three days would mean managing fewer active scheduling windows at any point in time. In addition, shortening the scheduling time is also desirable because high priority requests may arrive late, causing disruption to the overall schedule.

We decided that we could apply the empirical methodology we had developed for studying PFSP to this application. The goal is to help develop algorithms for assisting human schedulers in more efficiently building schedules that better utilize the available time slots.

4.1 The Satellite Scheduling Problem

Our research considers a simplified form of the satellite scheduling problem (SSP) that is ubiquitous in space-ground communications applications. The U.S. Air Force Satellite Control Network (AFSCN) is responsible for coordinating communications between users on the ground and satellites in space[1]. Communications to more than 100 satellites are performed through

[1]We would like to thank Alex Kilpatrick of AFOSR and Brian Bayless of Schriever Air Force Base for providing us with information on the application.

nine ground stations located around the globe, with an aggregate of sixteen antennas. To reserve a particular ground station for a period of time, users submit a task request which includes a required duration and a time window within which the duration must be allocated. The AFSCN scheduling center receives requests from a variety of users, attempting to satisfy the requests subject to the user-specified constraints. Over 500 requests are typically received for a single day.

Naturally, more requests are typically received than can be completely accommodated. After human schedulers attempt to fit all 500 or so tasks into the schedule, often about 120 conflicts, or requests that could not be accommodated, remain. Because satellites are extremely expensive resources, absolute rejection of requests is not an option. Rather, human schedulers must engage in a complex, time-consuming arbitration process to create a conflict-free schedule that maximizes utilization of the satellites.

Human schedulers use and balance many (potentially hard to quantify) criteria to develop a conflict-free schedule. In this study, we focused on a single, although crucial, aspect of the problem: minimizing the number conflicts before the arbitration process. Reducing the number of conflicts up-front reduces 1) the work-load of human schedulers, 2) communication with outside agencies, and 3) the time required to produce a feasible schedule.

4.2 Scheduling Satellite Access Requests

The SSP problem can be expressed as a decision problem: given a set of resources and a set of requests, does a feasible schedule exist? Often, demand exceeds the available capacity and the question then becomes how can one satisfy the maximum number of requests or otherwise best utilize the resources. Thus, we decomposed the SSP resource allocation problem into a two level scheduling problem. The higher-level problem is a **subset selection problem**: given the complete set of tasks, find the largest subset of tasks which can be feasibly scheduled. The lower-level problem is **deciding feasibility**: for a given subset of tasks does a feasible schedule exist? Both the high-level subset selection problem and the low-level feasibility decision problem are NP-Complete.

Because it is a decision problem, constraint-based scheduling is a natural framework for deciding SSP feasibility. Two heuristic approaches to constraint-based scheduling are prevalent in the literature: slack-based [19] and texture-based [2]. The texture-based approach advocates expending significant effort at each decision point to accurately characterize resource contention. High-contention points are identified, and the contributing tasks are scheduled to reduce the contention. In contrast, the slack-based approach advocates using inexpensive local information to achieve the same goals. To date, comparative studies show no clear advantage to either approach [2]. Both heuristics were originally applied to the static job-shop scheduling problem (JSP), and have subsequently been applied to extensions of JSP such as multiple capacitated resource scheduling [5] and the handling of alternative process plans [1]. We directly apply both the B-Slack [19] and Sum-Height [2] heuristics to deciding SSP feasibility.

Following [19] and [2], both heuristics are incorporated into the high-level search pseudocode ("the Basic Scheduling Algorithm") of [2]. Temporal arc-B consistency [12] is used when any ordering decision is posted, while Constraint-based Analysis (CBA) is used to detect both infeasible and implied/forced value orderings. However, we do *not* use the more powerful edge-finding Exclusion and Not-First/Not-Last propagators [14] employed in [2]. As discussed in Experiment 2 and 3 below, we allocate only a relatively small number of evaluations to determine schedule feasibility. Under these circumstances, we found the edge-finding propagators to actually impair performance.

In addition to chronological backtracking (CB), we examined three search algorithms from our previous study of PFSP (see Section 2.2): LDS, DDS, and HBSS. HBSS uses a bias function to determine next move; in all of our experiments, the quadratic bias function [3] yielded the best overall performance.

4.3 SSP Test Problem Generation

We designed a problem generator to mimic features found in the real-world problem. Tunable parameters of the generator allow the structure of the problems to be systematically varied. A problem consists of a set of task requests, each of which specifies a resource, a minimum duration, a maximum duration, and a time window specified in a (midpoint,width) form within which the task must be scheduled. To generate requests, the following parameters must be specified:

- the lower (D_l) and upper (D_u) bounds on the minimum task duration d_{min}
- the maximum duration multiplier (M_d), which implicitly limits the maximum task duration d_{max}
- the interval width multiplier (M_w), which limits the width of a task's scheduling time window

The minimum task durations d_{min} are sampled uniformly from the interval $[D_l, D_u]$. Once selected, the maximum duration is given by $d_{max} = d_{min} \cdot K$, where K is a constant uniformly sampled from the interval $[1.0, M_d]$. The time window midpoint W_m is sampled uniformly from the interval $[1, 1440]$, representing the length of a day in minutes. The time window width, W_w, is given by $W_w = d_{max} \cdot C$, where C is a constant uniformly sampled from the interval $[1.0, M_w]$. In our experiments, task durations are taken as d_{min}. Finally, because resources can be independently scheduled, we consider only single-resource problems.

4.4 Experiment 1: Phase Transitions and Search Costs for the SSP

Our algorithm for solving the SSP is a two-level search process. The low-level process decides feasibility of an SSP, and if that fails, the high-level process determines which tasks should be eliminated to produce a feasible schedule. Clearly, there is a trade-off when allocating CPU time to decide feasibility of sub-problems. If provided with inaccurate information regarding SSP feasibility, the high-level search process may schedule fewer than the maximum number of possible tasks. However, expending significant effort to decide feasibility will also limit the amount of search done by the high-level process, again possibly resulting in sub-optimal schedules.

To understand these tradeoffs, we need to identify interesting problem instances. Deciding feasibility for both under and over-constrained SSP instances is relatively inexpensive: most task ordering decisions are forced, and the heuristics are rarely employed. Further, by generating larger maximum duration and interval width multipliers (M_d and M_w) we can make the problems more difficult: increasing the scheduling window of each task increases both 1) the number of interactions with other tasks and 2) the number of possible scheduling alternatives. Thus, we expect the most difficult decision problems will possess a moderate number of tasks, each with a relatively large scheduling window. Problems with small M_d and M_w should be highly constrained, making feasibility easy to determine.

To confirm these hypotheses, we considered the following four problem classes, represented as the following M_d-M_w pairs: 2-2, 2-4, 4-4, and 4-6. For each problem class, $D_l = 10$ and $D_u = 40$. Problem size was varied from 2 to 100 tasks, in increments of 2, and 30 samples were generated for each problem size. For each heuristic/search algorithm combination, the following performance attributes were recorded on a per-instance basis: feasibility of the problem instance, number of evaluations (any node in the search tree) required, number of heuristic commitments made (binary nodes in the search tree), and, for soluble instances, the number of discrepancies required to reach the goal state.

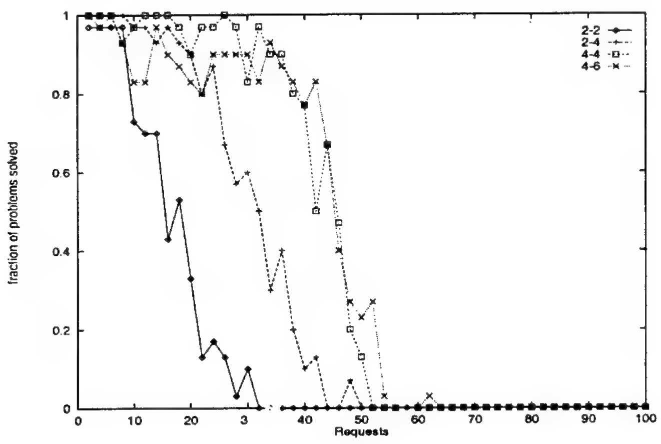
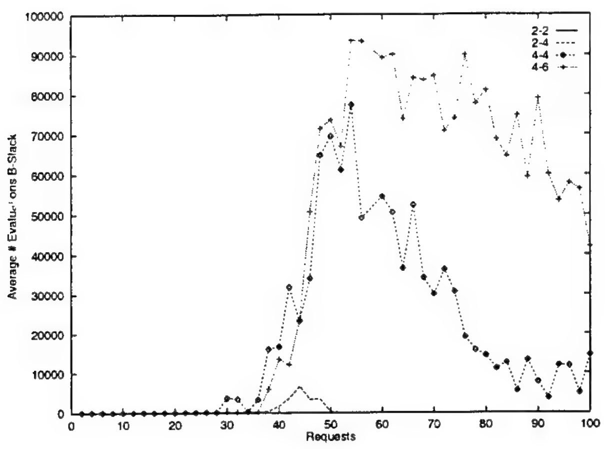


Figure 6: Fraction of feasible instances found for the four problem classes using B-Slack with CB.

We next evaluated B-Slack and Sum-Height in conjunction with chronological backtracking (CB). For each run, if 100K evaluations were consumed, search was terminated, and the problem was deemed infeasible. Figure 6 shows the fraction of feasible instances found with the B-Slack heuristic: the Sum-Height (not shown) results are nearly identical. Two features of Figure 6 are worth noting. First, we see a relatively abrupt transition between feasibility and infeasibility, as has been observed for SAT [11] and other NP-Complete problems [6]. Second, the slope of the feasibility-infeasibility transition is relatively independent of the problem class.

Figure 7 shows the search cost, within the 100K evaluation limit, required to decide feasibility of the problem instances used to produce Figure 6. Here, we see two main effects. First, the peak in computation cost slightly lags behind the transition in feasibility. Further, the computation cost tends to gradually decay after the phase transition, i.e., it is still relatively costly to determine solubility for some infeasible problem instances. Second, as we hypothesized,



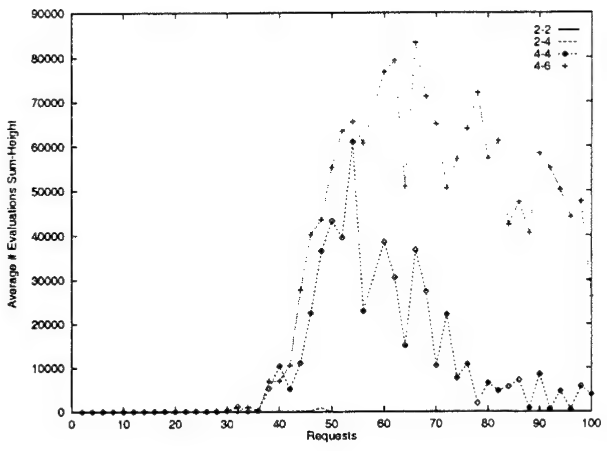


Figure 7: Leftmost graph: Average number of evaluations required to decide feasibility, using B-Slack with Chronological Backtracking. Rightmost graph: Average number of evaluations required to decide feasibility, using Sum-Height with Chronological Backtracking.

problem difficulty increases with increases in the size of task scheduling windows – determining feasibility is relatively easy for the 2-2 and 2-4 problem classes. Although not shown, the number of discrepancies and heuristic commitments also peak just after the transition region. To summarize: the most difficult and challenging problems are those near the phase transition and that possess relatively large task scheduling windows.

4.5 Experiment 2: Selecting a Feasibility Search Algorithm

Next, we characterized the performance of each heuristic under the systematic search algorithms with the goal of determining the best cost vs. probability of solution tradeoff for the low-level search algorithm. Empirically, proving infeasibility typically requires an excessive ($> 30K$) number of evaluations. As shown below, proving feasibility is often far more tractable. Therefore, because we can only afford to allocate a small number of evaluations to decide feasibility, we only consider *feasible* problem instances below.

Figure 8 shows the percentage feasible problems (those constructed in Experiment 1) solved by LDS and HBSS within 100K evaluations. First, we note that LDS always outperforms HBSS, independent of heuristic. Empirically, relatively few discrepancies ($<= 8$) are required to decide feasibility for most of these instances. This is an advantage for LDS, which always finds solutions with the smallest number of discrepancies possible. DDS under-performed both HBSS and LDS, because the heuristic discrepancies required to establish feasibility often occurred deep in the search tree. Second, for tasks with M_d-M_w duration pairs 2-2, 2-4, 4-4, the performance of both slack-based and texture-based heuristics is almost identical under LDS (only 4-4 is shown in Figure 8a). Third, when using LDS, the B-Slack heuristic performs slightly better than Sum-Height on the more difficult 4-6 M_d-M_w instances.

Figure 8b shows the average number of evaluations needed to solve feasible problems, within the 100K limit. B-Slack requires more evaluations, but Sum-Height is more computationally intensive. For a single resource, B-Slack and Sum-Height are, in the worst case, $O(n^2)$ and

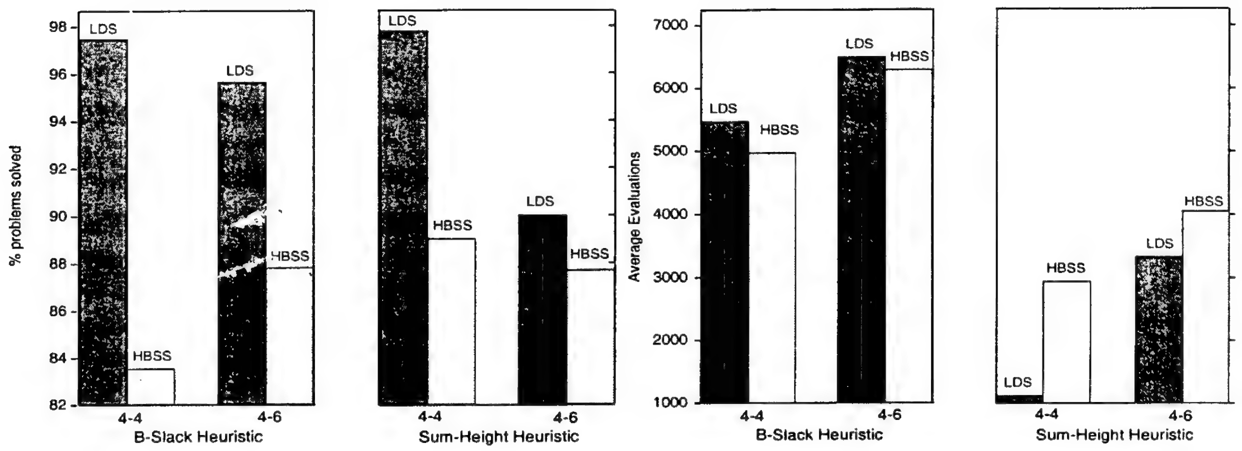


Figure 8: The two leftmost graphs show the percentage of feasible instances solved by B-Slack and Sum-Height, using LDS and HBSS. The two rightmost graphs show the average number of evaluations to solve feasible instance solved by B-Slack and Sum-Height, using LDS and HBSS.

$O(n^2) + O(nlog(n)) + O(n)$, respectively; n is the number of tasks assigned to the resource. Our implementation of Sum-Height is 4 times more costly to compute than B-Slack. Given this factor, Sum-Height does less work on the 4-4 problems, but does more total work on the 4-6 problems. However, we found the total execution times too similar to claim that one heuristic was faster than another for deciding feasibility. But since B-Slack was equal to Sum-Height on 2-2, 2-4 and 4-4 problems, and resulted in more problems solved on difficult 4-6 problems, we elected to use B-Slack to decide feasibility of SSP instances.

4.6 Experiment 3: Selecting a Subset Selection Algorithm

Slack-based and texture-based heuristics both operate by identifying areas of schedule contention, and posting task ordering decisions to reduce the contention. The slack-based approach assumes that such information can be obtained by local examination of task interactions, while the texture-based approach assumes a more detailed, global view is required. Contention is also important in determining which tasks to remove, or *bump*, from the schedule to establish feasibility. In an over-constrained schedule, eliminating the largest contributors to high-contention regions of the schedule is likely to move the schedule toward feasibility.

Because it provides a global view of the contention, our texture-based bump heuristic (T-Bump) is a straightforward extension of the Sum-Height heuristic: identify the region of the schedule with the largest contention, and bump the task which contributes most. In contrast, we found that strictly local examination of slack between pairs of tasks was an extremely poor heuristic. Thus, we defined a new slack-based bump heuristic (S-Bump) as follows. Consider a task i that interacts (via overlaps in their scheduling windows) with N other tasks j through ordering decisions O_{ij}, $i \neq j$. Define $\overline{slack_i} = \sum_{j=1}^{N} max(bslack(i \rightarrow j), bslack(j \rightarrow i))/N$. Small $\overline{slack_i}$ values are used to identify highly constrained tasks i; posting further constraints is likely to lead to infeasibilities. The S-Bump heuristic operates by selecting the task i such that $\overline{slack_i}$ is minimized.

Class	Heuristic	40	45	50	55	60	65	70
2-4	S-Bump	0	0	3	3			
	T-Bump	0	1	-1	1			
4-4	S-Bump		1	2	2	1	3	
	T-Bump		0	0	3	4	4	
4-6	S-Bump			-1	8	6	4	10
	T-Bump			1	3	9	5	8

Table 1: The difference in number of scheduled requests under LDS and HBSS for both S-Bump and T-Bump heuristics. A positive number indicates LDS scheduled more requests; negative numbers show HBSS was best.

Both S-Bump and T-Bump select a single task to be removed from the schedule. The alternative choice is to force task inclusion into the schedule. Before computing either heuristic, all active tasks are placed on the schedule and edge-finding (both Exclusion and Not-First/Not-Last forms) [14] is applied to constrain the initial placement of each task by taking into account task interactions. Edge-finding examines subsets of tasks to determine constraints on their scheduling windows. Without edge-finding, S-Bump performed very poorly and the performance of T-Bump was only slightly degraded. Finally, we considered both heuristics in conjunction with heuristic search algorithms.

To test our bump heuristics, we considered test problems that are between the 50% point in the phase transition and the point where all problems are insoluble. For the problem classes 2-2, 2-4, 4-4, and 4-6 the number of tasks ranged from 20-40, 35-55, 45-65, and 50-70, respectively, in increments of 5. For each problem size, 10 test problems were generated. We then ran each combination of bump heuristic (S-Bump or T-Bump) and search algorithm (LDS, DDS, or HBSS) and recorded the total number of tasks that were eliminated in order to produce a feasible schedule. Each run of the high-level task subset selection algorithm was allocated a maximum of 10K evaluations. For each candidate solution generated by the high-level algorithm, we used B-Slack and LDS without edge-finding to decide schedule feasibility. Based on Figure 8b, we allocated 1K for 2-2/2-4 problems and 8K evaluations for 4-4/4-6 problems.

Table 1 shows the *difference* in the number of conflicts (bumped tasks) between LDS and HBSS; the reported value is the aggregate over all 10 problem instances in the category. Positive values indicate LDS produced schedules with fewer conflicts than HBSS. Problem class 2-2 showed no differences in performance between LDS and HBSS. Here, smaller task scheduling windows allow optimal solutions to be quickly found. For the problem classes with larger task scheduling windows, LDS significantly outperforms HBSS. The difference is negligible for problems near the 50% point of the phase transition, but is considerable for problems in the insoluble region.

The poor performance of HBSS is partially attributable to the use of a fixed bias function. Under the quadratic bias function (and a binary search space), both heuristics are followed with probability 0.80. Given a typical search tree depth of 60, the expected number of discrepancies on any root-to-leaf path is 12; this is larger than the number of discrepancies required by LDS (which averages 5 and is never more than 8). Thus, careful tuning of HBSS is required to

Problem Class	45	50	55	60	65	70
2-4	0	7	3	-	-	-
4-4	9	2	6	11	13	-
4-6	-	13	7	16	12	22

Table 2: The difference in number of conflicts between the S-Bump and T-Bump heuristics, under LDS. A positive value indicates T-Bump was best.

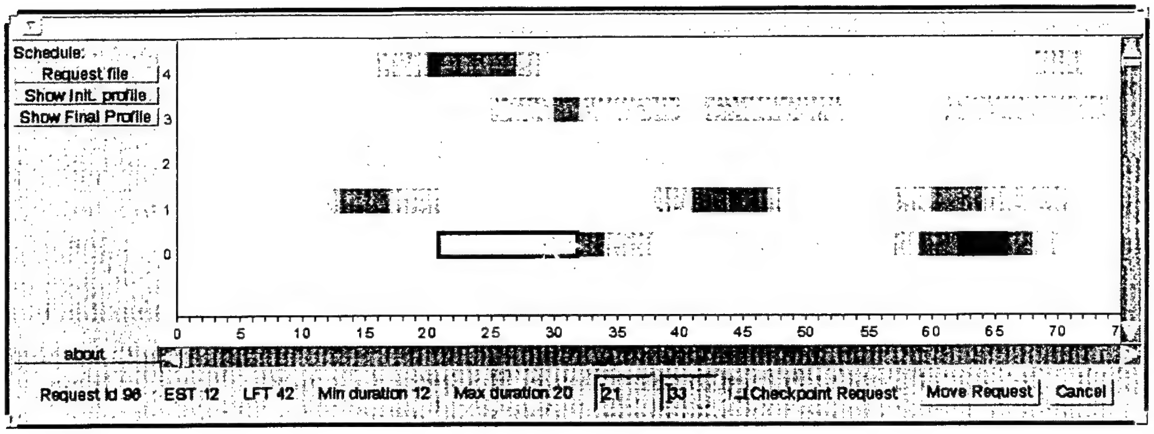
achieve good performance–both search tree depth and heuristic quality should be taken into account. A more promising approach is to dynamically compute the bias, as explored in [15]. Finally, DDS under-performed both LDS and HBSS because discrepancies deep in the search tree were required.

Next, we considered the relative performance of S-Bump and T-Bump heuristics by comparing the performance of each in conjunction with LDS. Table 2 shows the *difference* in the number of conflicts between S-Bump and T-Bump; reported values are aggregates over all 10 problem instances in the category. Positive values indicate T-Bump produced schedules with fewer conflicts than S-Bump. We saw no performance difference on problem class 2-2: such instances are over-constrained and easy to solve. However, T-Bump significantly outperforms S-Bump on the more under-constrained problem classes. Further, the differences become larger with increases in 1) the task scheduling windows and 2) the distance from the 50% point of the phase transition. We hypothesized that characterizing contention is key to determining which tasks should be bumped from the schedule. Based on our experimental results, we conjecture that slack-based heuristics can accurately characterize global contention for moderately over-constrained problems such as the JSP. However, the approach begins to break down in heavily over-constrained problems such as satellite scheduling.

4.7 Interface to AFSCN Schedules

We developed a simple Java interface to the schedules. The purpose of the interface is to allow a user to visualize conflicts and consider alternatives for dealing with them. Figure 9 illustrates a hypothetical situation where the initial set of requests has been displayed with darker levels of gray-scale indicating greater conflict (this would be in color online). The user can examine the set and then select a specific request to be "checkpointed." The dark rectangle outline highlights a specific client request. If this particular request appears to represent a significant bottleneck, the request can be checkpointed to allow the user to look at the partial schedule as it exists at the point when the system is trying to schedule this particular request.

At present, the system generates initial schedules using min-slack/greedy search and allows the user to view, add, delete or move requests within the proposed schedule. We chose Java so that it could be made available to users and experts over the World Wide Web. We intend in a follow-up project to considerably extend the capabilities of the prototype.



Request Id 98 EST 12 LFT 42 Min duration 12 Max duration 20 21 33

Figure 9: User can view a schedule to identify bottlenecks and requests that may need to be negotiated.

5 Accomplishments/New Findings

Flow Shop Scheduling Results: PFSP algorithms have been declared to be state of the art when they out-perform others on known test problem sets, such as the Taillard problems. The advantage of these problem sets is that they allow comparisons with existing methods via published results. However, a common complaint is that methods that work well on test problems often do not work well in real-world domains. If this is true, then applying a method based on published accounts is a questionable endeavor; excellent published performance may not generalize.

We developed a methodology for testing scheduling algorithms and explaining aspects of the performance. We built a problem generator that introduces tunable amounts and types of structure into PFSP problems. We then tested a suite of state-of-the-art algorithms on problems with carefully controlled amounts and type of structure. From their observed performance, we determined whether the method's performance generalized. We followed up the performance assessments by examining the search space for factors that would explain observed departures in performance.

We have found that correlated job sizes differentially affect the performance of algorithms. We have found, in general that simple stochastic algorithms using restarts (e.g., the Heuristic Bias Stochastic Sampling algorithm, HBSS) with a good heuristic (in this case, NEH, a 1983 algorithm from the OR literature) produces the best overall results, and also are the most robust with respect to changes in problem structure and scale-up. Stochastic algorithms out-performed systematic algorithms. Additionally, branch and bound algorithms are no longer as effective in either finding or confirming optimal solutions because the lower bound is so far removed from the optimal solution.

On job-correlated problems, path relinking is the only algorithm to sometimes out-perform the random restarts or HBSS with NEH algorithm–often the algorithms produce the same answers. Path relinking also seems to perform well independent of the level of problem structure. However, NEH is cheaper to execute, and path relinking performs relatively poorly on machine-correlated problems.

These results show that exploiting even very simple problem structure (such as these correlations) can significantly improve performance in real world domains. We have also found that most systematic and stochastic algorithms can be improved by hybridizing the algorithm to include the occasional application of local search. Algorithms such as tabu search and simulated annealing automatically include local search, but other methods do not.

We also partly explain how introducing correlation into a problem impacts the structure of the search space and the difficulty of problems. Random problems have local optima that are generally grouped together, where local optima that are "near" to each other tend to have a similar evaluation. But otherwise, these local optima tend to have distinct evaluations and little other structure that is easy to characterize. However, as soon as processing times become correlated (either in terms of machine processing time, or with respect to the processing time of specific jobs), the local optima become organized into a plateau structure. The plateaus overlap, but in general, the steps that are closer to the best known local optima have better evaluations. Optima on the same plateau have the same evaluation–and also tend to be linked in the sense that one can move from one solution to another on a plateau without changing the evaluation. If one views all local optima on the same plateau as the same optimum, then the introduction of problem structure in the form of correlation reduces the number of local optima compared to random problems.

Our work has produced two major results. First, we demonstrate that superior performance on a popular synthetic benchmark test suite fails to transfer to a test suite containing problems with only modest levels of non-random features. Thus, our new test problems should also have a significant impact on how researchers evaluate and test their scheduling systems. Second, we show that the non-random flow-shop problems do not display a search space topology previously associated with difficult problems, thus partially explaining why previously successful algorithms which exploit such structure fail on non-random problems.

Theoretical Results: Using a local search operator over a neighborhood of k points, we can calculate the probability that any particular point in the search space will be a local minima under an arbitrary change in representation. We also now have a closed form solution that gives the expected number of local minima over all possible problems/representation.

A sub-theme to our work has related to "No-Free-Lunch" results: that is, over all possible discrete problems, all algorithms have the same performance on average. We have been able to generate new theoretical results concerning when No-Free-Lunch results are interesting–and when they are not. First, No-Free-Lunch proves that black-box optimization, in the most general case, does not work. On the other hand, we have proven that 1) for all known No-Free-Lunch proofs, the set of problems that are covered by No-Free-Lunch on average have exponential description length; hence, in some sense these problems "on average" do not correspond to problems of practical interest. On the other hand, we have proven that 2) "Free-Lunch" proofs can be constructed that allow one to pick one representation over another. Gray codes and Binary codes for discretized parameter optimization are the same over all problems, but on certain classes of problems with polynomial description Gray codes are superior to Binary codes in the sense that Gray code representations, on average, induces fewer optima. It would be extremely interesting if similar results could be developed for permutation based representations for scheduling problems, but it is not clear how to proceed toward such a theory. Hence we have devoted the vast majority of our time and attention toward understanding the specific kinds of problem structure that exists for specific applications such as flowshop, jobshop and Remote Ground Station Scheduling.

Application Results: The methodology we developed for identifying good algorithms for PFSP was applied to the AFSCN application. We developed a problem generator for the application based on conversations with personnel at Schriever Air Force Base. We modified two state of the art job shop scheduling heuristics, minslack and texture, to serve this problem. Then we tested performance of algorithm/heuristic combinations on a suite of controlled problems from the generator.

We found that slack-based heuristics perform better for deciding schedule feasibility, while texture-based methods are better at deciding which tasks to eliminate from the schedule. As slack-based heuristics are considerably cheaper to compute than texture-based, more schedules can be assessed for feasibility in the same amount of time.

As with PFSP, we examined the search spaces for the problems. We documented the existence of a phase transition for the schedule feasibility decision problem. The problems exhibit a relatively rapid transition from feasibility to infeasibility as the number of tasks is increased, and a substantial increase in problem difficulty near the transition region.

To assist in the examination of proposed schedules, we built a Java interface to the schedules. The interface allows users to graph the schedules in different ways and observe the effect of changes.

6 Executive Summary

6.1 Personnel

During the grant period, the following personnel were supported at the indicated level:

PIs:			
	Adele Howe	4.0	months
	L. Darrell Whitley	4.0	months
Research Assistants:			
	Laura Barbulescu	20.5	months
	Sridhar Kandukuri	4.5	months
	Jean-Paul Watson	13.0	months
Consultant:			
	Fred Glover	12	days
Administrative Assistant:			
	Lisa Kennedy	10.0	months

6.2 Publications

- L. Barbulescu, J.P. Watson, A.E. Howe and L.D. Whitley. (1999) "Problem Structure and Flowshop Scheduling", *Proceedings of CEDYA 99*, Las Palmas de Gran Canaria.
- F. Glover.(1997) "A Template for Scatter Search and Path Relinking". *Artificial Evolution*", Lecture Notes in Computer Science, 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (Eds.) Springer, pp. 13-54.
- R. Heckendorn, S. Rana, D. Whitley. (1998) "Test functions generators as embedded landscapes". In *Foundations of Genetic Algorithms*, C. Reeves and W. Banzhaf, ed., Morgan Kauffman.
- R. Heckendorn, S. Rana, and D. Whitley (1999). "Polynomial time summary statistics for a generalization of MAXSAT". *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-99.* pp: 281-288.
- R. Heckendorn, D. Whitley. "Predicting epistasis directly from mathematical models". *Evolutionary Computation.* 7(1):69-101.
- A.E. Howe, L.D. Whitley, L. Barbulescu, J.P. Watson. (2000) Mixed Initiative Scheduling for the Air Force Satellite Control Network, *Second International NASA Workshop on Planning and Scheduling for Space*, March 2000.
- A.E. Howe, L.D. Whitley, J.P. Watson, L. Barbulescu.(2000) "A Study of Air Force Satellite Access Scheduling", In *Proceedings of the World Automation Conference*, Maui, HI, June 2000.
- S. Rana, R. Heckendorn, D. Whitley. (1998) "A Tractable Walsh Analysis of SAT and its Implications for Genetic Algorithms". *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-98).* pp. 392-397, July 1998.
- M. Vose, D. Whitley. (1998) "A Formal Language for Permutation Recombination Operators". In *Foundations of Genetic Algorithms*, C. Reeves and W. Banzhaf, ed., Morgan Kauffman.
- J.P. Watson, L. Barbulescu, A. Howe and D. Whitley. (1999). "Algorithm Performance and Problem Structure for Flow-Shop Scheduling". *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI 99).* Orlando, FL, July.
- J.P. Watson, S. Rana, D. Whitley and A. Howe (1999). "The Impact of Approximate Evaluation on the Performance of Search Algorithms for Warehouse Scheduling". *Journal on Scheduling*, 2(2):78-98.
- L. D. Whitley, A. E. Howe, S. Rana, J. P. Watson and L. Barbulescu. (1998) "Comparing Heuristic Search Methods and Genetic Algorithms for Warehouse Scheduling", In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Diego, CA. October, 1998.

- L. D. Whitley, S. Rana and R. Heckendorn (1999). "The Island Model Genetic Algorithm: On Separability, Population Size and Convergence". *Journal of Computer and Information Technology*, 7(1):33-47.
- L.D. Whitley (1999). "A Free-Lunch Proof for Gray versus Binary Encodings". *Genetic and Evolutionary Computation Conference, GECCO-99.* pp: 726-733.
- D. Whitley and L. Barbulescu and J.P. Watson (2001). Local Search and High Precision Gray Codes: Convergence Results and Neighborhoods. *Foundations of Genetic Algorithms, FOGA-6.* Morgan Kaufmann.
- M. Vazquez and D. Whitley (2000). A Comparison of algorithms for the Static Job Shop Problem. *Parallel Problem Solving from Nature.* Springer-Verlag.
- Manuel Vazquez and D. Whitley (2000). A Comparison of Genetic Algorithms for Dynamic Job Shop Scheduling. *Genetic and Evolutionary Computation Conference*, GECCO-2000.
- Manuel Vazquez and D. Whitley (2000). A Hybird Genetic Algorithm for the Quadratic Assignment Problem. *Genetic and Evolutionary Computation Conference*, GECCO-2000.
- D. Whitley (2000). Functions as Permutations: Implications for No Free Lunch, Walsh Analysis and Statistics. *Parallel Problem Solving from Nature.* Springer-Verlag.
- L. Barbulescu, J.P. Watson, and D. Whitley (2000). Dynamic Representations and Escaping Local Optima. *AAAI-2000.* pp. 879-884.
- J.P. Watson, L. Barbulescu, A.E. Howe and L.D. Whitley. (2000) "Contrasting Structured and Random Permutation Flow-Shop Scheduling Problems: Search Space Topology and Algorithm Performance", Submitted to *Journal of Computing.*

6.3 Graduate Theses

Both graduate research assistants completed their M.S. degrees within the grant period. Their theses addressed issues from the grant. These documents can be obtained from the authors or from Colorado State University.

"Lower Bound Computation for Structured Flowshop Scheduling" M.S. Thesis by Laura Barbulescu in Fall 1999. Abstract:

> In flowshop scheduling, lower bound computations can guide search and facilitate comparisons between solution methods. The methods used to compute the lower bound for flowshop problems exhibit a trade-off between the computational complexity and the sharpness of the produced lower bound. When information about the structure of the problem is available, simple methods can be designed that compute lower bound values of quality similar to the ones produced by computationally

intensive methods. I study two types of structure in flowshop scheduling by generating two classes of problems. For the first problem class, for each job, I generate processing times that are similar across all machines (job-correlated problems). For the second problem class, for each machine, I generate processing times that are similar for all the jobs (machine-correlated problems). For both types of problems I identify easy to compute, tight lower bounds and relate them to existing lower bound computations.

"The Impact of Approximate Evaluation on the Performance of Search Algorithms for Warehouse Scheduling" M.S. Thesis by Jean-Paul Watson in Fall 1999. Abstract:

The Coors warehouse scheduling problem involves finding a permutation of customer orders that minimizes the average time that customers' orders spend at the loading docks while at the same time minimizing the running average inventory. Search based solutions require fast objective functions. Thus, a fast low-resolution simulation is used as an objective function. A slower high-resolution simulation is used to validate solutions. We compare the performance of a constructive scheduling algorithm to a genetic algorithm and local search approach. The constructive algorithm is based on a heuristic built specifically for this application. We also tested a hybrid of the genetic algorithm and local search approaches by initializing the search using the domain-specific heuristic. This hybrid genetic algorithm was able to find the best solutions when evaluated by the high-resolution simulation. Finally, we consider the effect of using the high-resolution simulation to filter a set of solutions found by the different approaches.

6.4 Other Products

Software As part of this project, we have developed two problem generators and a user interface for the AFSCN application. We have made our PFSP problem generator publically available via a web site for other researchers. Our problem generator for the AFSCN problem can be made available by request.

We have also developed a prototype user interface in Java for the AFSCN scheduling problem. The interface allows a user to inspect proposed schedules and make minor changes to them.

Web Site Our project web site is available at `http://www.cs.colostate.edu/sched/`. From that site, you can access publications, software and problem instances from the project. By making the problems and the generator available, we provide an alternative to the Taillard benchmarks.

References

[1] J. Christopher Beck. Scheduling alternate activities. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, Orlando, FL, 1999.

[2] J. Christopher Beck, Andrew J. Davenport, Edward M. Sitarski, and Mark S. Fox. Texture-based heuristic for scheduling revisited. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, 1997.

[3] John L. Bresina. Heuristic-biased stochastic sampling. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996.

[4] R. Caruana and J. Schaffer. Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms. In *Proc. of the 5th Int'l. Conf. on Machine Learning.* Morgan Kaufmann, 1988.

[5] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In *Proc. of the Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

[6] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[7] J. Culberson. On the Futility of Blind Search. *Evolutionary Computation*, 6(2):109–127, 1999.

[8] Michael R. Garey and David S. Johnson. *Computers And Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freemand and Company, 1979.

[9] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

[10] S.A. Kauffman. Adaptation on Rugged Fitness Landscapes. In D.L. Stein, editor, *Lectures in the Science of Complexity*, pages 527–618. Addison-Wesley, 1989.

[11] S. Kirkpartrick and B. Selman. Critical behavior in the satisfiability of boolean expressions. *Science*, 264:1297–1301, 1994.

[12] O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC, 1993.

[13] M. Nawaz, E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11/1:91–95, 1983.

[14] W. P. M. Nuijten. *Time and Resource Constrained Scheduling: a Constraint Satisfaction Approach.* PhD thesis, Eindhoven University of Technology, 1994.

[15] Angelo Oddi and Stephen F. Smith. Stochastic procedures for generating feasible schedules. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, RI, 1997.

[16] N.J. Radcliffe and P.D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In J. van Leeuwen, editor, *Lecture Notes in Computer Science 1000.* Springer-Verlag, 1995.

[17] S. Rana and D. Whitley. Search, representation and counting optima. In L. Davis, K. De Jong, M. Vose, and D. Whitley, editors, *Proc IMA Workshop on Evolutionary Algorithms.* Springer-Verlag, 1998.

[18] Colin R. Reeves and Takeshi Yamada. Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6:45–60, 1998.

[19] S.F. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC, 1993.

[20] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operations Research*, 47:65–74, 1990.

[21] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.

[22] Craig A. Tovey. Hill climbing and multiple local optima. *SIAM Journal on Algebraic and Discrete Methods*, 6(3):384–393, July 1985.

[23] Toby Walsh. Depth-bounded discrepancy search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR, 1996.

[24] D. Whitley. A Free Lunch Proof for Gray versus Binary Encodings. In *GECCO-99*, pages 726–733. Morgan Kaufmann, 1999.

[25] Darrell Whitley and Soraya Rana. Representation, search and genetic algorithms. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

[26] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.